

CSCI 3753: Operating Systems

Week 3
September 5, 2025





University of Colorado **Boulder**

Announcements

- Quiz 2 due tonight at midnight!
- PA1 is due Sunday at midnight!
- PA2 and PA3 are SIGNIFICANTLY MORE WORK THAN PA0 and PA1
- Interview grading is NEXT WEEK
 - Schedule your place TODAY on Canvas Calendar
- Recitation materials: <https://tinyurl.com/CSCI3753>

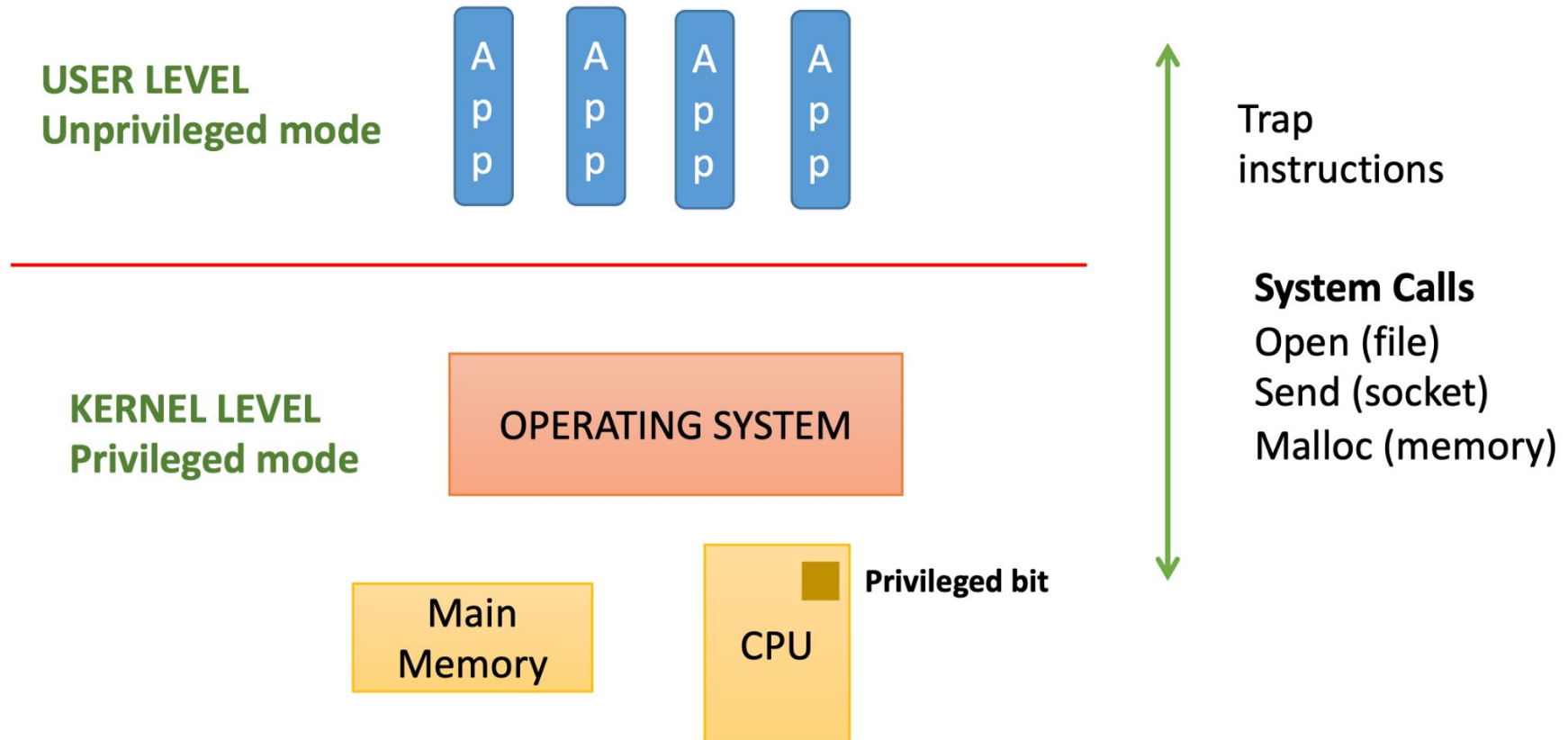


Recap

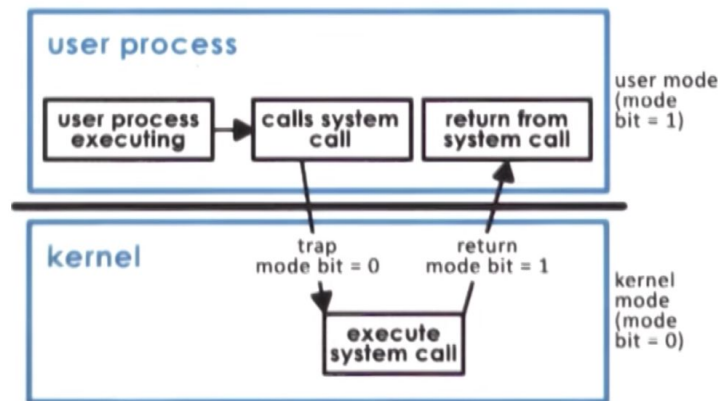
- Bootloader
- User/Kernel space
- System calls
 - Trap tables
- PA0 
- PA1 



User/Kernel Protected Boundary



System Call Flow



- To make a system call a program must:
 - write arguments
 - save relevant data to a defined location
 - make call using the specific system call number

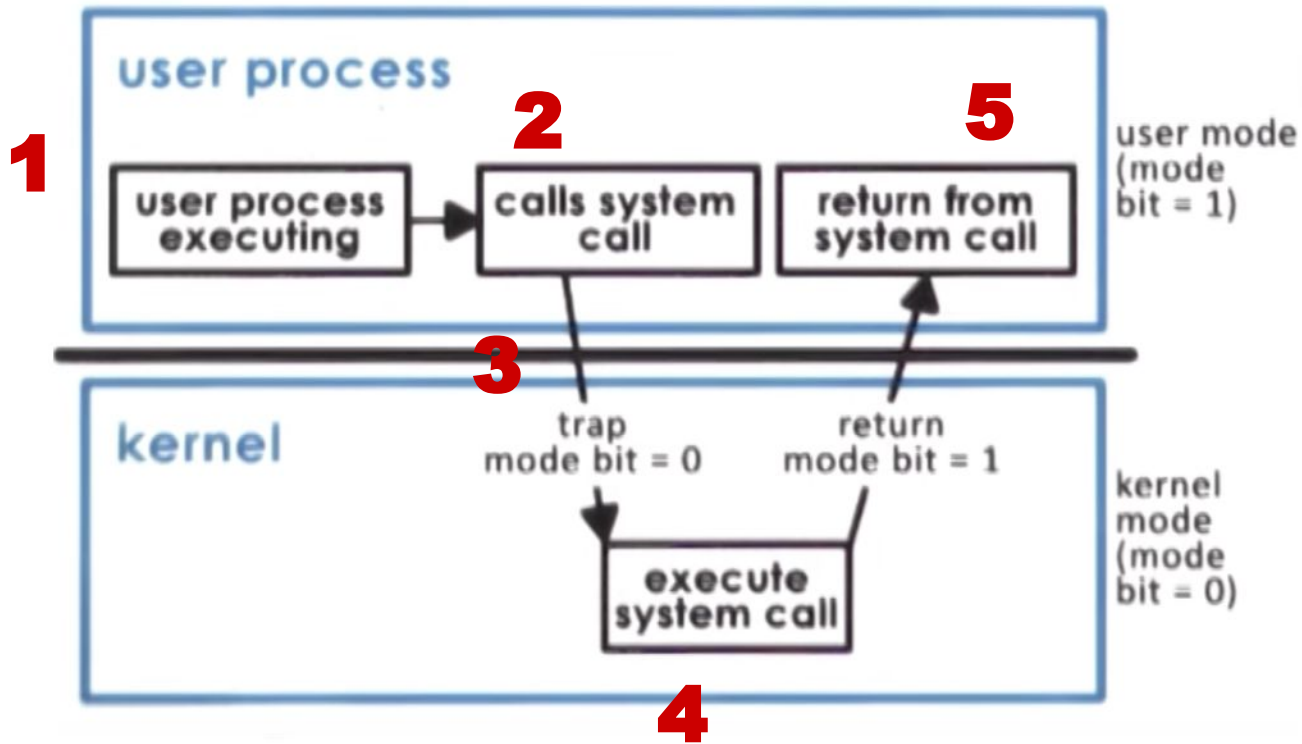


Executing System Call

1. A user space program invokes the syscall
2. A (usually) software interrupt is called and a trap is triggered (INT)
3. Mode bit is flipped from user to kernel (1 to 0)
4. The interrupt informs the kernel which syscall is needed
 - a. Requisite data is retrieved
 - b. Kernel verifies if all parameters are valid before executing the syscall
5. After execution, mode bit flips and user program resume



System Call Flow



Adding A System Call

1. Write the system call source code
2. Add the new syscall to the Makefile
3. Add the syscall to the syscalls table
4. Add the syscall prototype to the syscalls header file
5. Recompile, install, boot into mod'd kernel



Adding A System Call

1. Write the system call source code

```
#include <linux/kernel.h>
#include <linux/linkage.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE0(helloworld)
{
    printk(KERN_ALERT "hello world\n");
    return 0;
}
```

1. Add the new syscall to the Makefile
2. Add the syscall to the syscalls table
3. Add the syscall prototype to the syscalls header file
4. Recompile, install, boot into mod'd kernel



Adding A System Call

1. Write the system call source code
2. Add the new syscall to the Makefile
 - a. within './arch/x86/kernel/Makefile' add the line:

```
obj-y+=helloworld.o
```

1. Add the syscall to the syscalls table
2. Add the syscall prototype to the syscalls header file
3. Recompile, install, boot into mod'd kernel



Adding A System Call

1. Write the system call source code
2. Add the new syscall to the Makefile
3. Add the syscall to the syscalls table
 - a. Within file './arch/x86/entry/syscalls/syscall_64.tbl' add:

```
447      common helloworld          sys_helloworld
```

1. Add the syscall prototype to the syscalls header file
2. Recompile, install, boot into mod'd kernel



Adding A System Call

1. Write the system call source code
2. Add the new syscall to the Makefile
3. Add the syscall to the syscalls table
4. Add the syscall prototype to the syscalls header file
 - a. Within the file './include/linux/syscalls.h' add the line:

```
asmlinkage long sys_helloworld(void);
```

1. Recompile, install, boot into mod'd kernel



Adding A System Call

1. Write the system call source code
2. Add the new syscall to the Makefile
3. Add the syscall to the syscalls table
4. Add the syscall prototype to the syscalls header file
5. Recompile, install, boot into mod'd kernel

```
cd /home/kernel/linux-hwe-5.13-5.13.0  
make -j10  
sudo make modules_install  
sudo make install  
sudo reboot now
```



Loadable Kernel Module (LKM)

- LKM is a chunk of code we add to the Linux kernel while it is running
- Most often we do this with these modules:
 - Device drivers
 - Filesystem drivers
 - System calls



Loadable Kernel Module (LKM)

- LKMs are part of the kernel, once loaded
 - The part of the kernel bound to the machine image on boot is “base kernel”
- LKMs communicate with the base kernel



How to Add LKM

1. Write LKM code
2. Add module to Makefile
3. Compile the module to get the .o file
4. Insert the mod into the running kernel



How to Add LKM

1. Write LKM code

Create a directory, `/home/kernel/modules`, and edit a new file named `helloModule.c`. Populate this file with the following code:

```
#include<linux/init.h>
#include<linux/module.h>

MODULE_AUTHOR("Your Name");
MODULE_LICENSE("GPL");

int hello_init(void) {
    printk(KERN_ALERT "inside %s function\n",__FUNCTION__);
    return 0;
}

void hello_exit(void) {
    printk(KERN_ALERT "inside %s function\n",__FUNCTION__);
}

module_init(hello_init);
module_exit(hello_exit);
```

1. Add module to Makefile
2. Compile the module to get the .o file
3. Insert the mod into the running kernel



How to Add LKM

1. Write LKM code
2. Add module to Makefile

```
obj-m:=helloModule.o
```

1. Compile the module to get the .o file
2. Insert the mod into the running kernel



How to Add LKM

1. Write LKM code
2. Add module to Makefile
3. Compile the module to get the .o file

```
make -C /lib/modules/$(uname -r)/build M=$PWD
```

1. Insert the mod into the running kernel



How to Add LKM

1. Write LKM code
2. Add module to Makefile
3. Compile the module to get the .o file
4. Insert the mod into the running kernel

```
user@csci3753:/home/kernel/modules$ sudo insmod helloModule.ko  
user@csci3753:/home/kernel/modules$ lsmod | grep hello  
helloModule 16384 0
```

Much faster!



Why Use LKMs?

- We DON'T have to rebuild the kernel
- Help diagnose system problems
 - A device driver bug could bring down the kernel/entire system
- Save memory
 - Only load kernels when necessary
- Much faster to maintain, debug, and deploy



LKM Utilities

- insmod - insert LKM into kernel
- rmmod - remove LKM from kernel
- lsmod - list currently loaded LKMs
- kernels - kernel daemon program (for automation)
- modprobe - ins/rm one or multiple LKMs intelligently
 - e.g., if mod A must be loaded before mod B, modprobe will automatically load mod A if you request mod B
- depmod - determine mod interdependency
- ksyms - display symbols exported for the new LKM
- modinfo - displays LKM info from .modinfo section of LKM obj file



Passing Data Safely Between User/Kernel Space

- `copy_from_user`
- `copy_to_user`
- `strnlen_user`
- `strncpy_from_user`
- `get_user`
- `put_user`

Where are we getting this from? [Kernel Memory Access](#) PDF on Canvas



PA1

- What questions do you all have?
- How many people have already finished?
- What has been the hardest part to complete?

